VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webová aplikace pro tvorbu třídních UML diagramů

Web Application for Creation of Class Diagrams

2016 Tomáš Řeha

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Řeha**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Webová aplikace pro tvorbu třídních UML diagramů
Web Application for Creation of Class Diagrams

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je implementace aplikace pro tvorbu třídních diagramů a generování entitní a databázové vrstvy. Student má za úkol prostudování problematiky tvorby třídních UML diagramů a možností jejich převodu na zdrojový kód. Na základě nabytých znalostí pak implementuje navrženou aplikaci pro tvorbu třídních diagramů a generování zdrojových kódů.

Práce bude obsahovat:
1. Popis problematiky tvorby třídních diagramů, převodu na zdrojový kód a možnosti generování databázového modelu.
2. Analýzu a návrh vlastní aplikace, která bude umožnovat tvorbu třídních UML diagramů a také generování entitní a databázové vrstvy.
3. Implementaci navržené aplikace.
4. Uživatelskou a programátorskou dokumentaci.

Seznam doporučené odborné literatury:

[1] Vondrák I., Kožusznik J., Ochodková E.: Metody specifikace softwarových systémů. Katedra Informatiky FEI, VŠB-TU Ostrava, 2006.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radoslav Štrba**

Datum zadání: 01.09.2015
Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 29. dubna 2016 .....

Na tomto místě bych chtěl poděkovat Ing. Radoslavu Štrbovi za cenné rady a odborný přístup při zpracovávání této bakalářské práce.

Abstrakt

Hlavním cílem této bakalářské práce bylo analyzovat, navrhnout a implementovat webovou aplikaci, která umožní vytváření třídních diagramů s možností generování zdrojových kódů. Aplikace umožňuje vygenerovat SQL script a odpovídající zdrojové kódy v C# jazyce. Zdrojové kódy jsou rozděleny do entitní a datové vrstvy. Datová vrstva slouží pro komunikaci s danou databází. Implementace je provedena pomocí moderních internetových technologií HTML5, CSS3 a JavaScriptu.

Klíčová slova: JavaScript, HTML5, Třídní diagram, SQL, C#

Abstract

The main goal of this bachelor thesis was to analyze, design and implement a web application that allows you to create class diagrams with the possibility of generating source codes. This application allows you to generate SQL script and the corresponding source code in C# programming language. The source code is divided into two layers of an application, the entity layer and the data layer. The data layer is used to communicate with a database. Modern internet technologies such as HTML5, CSS3 and JavaScript were used for implementation.

Key Words: JavaScript, HTML5, Class diagram, SQL, C#

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CSS3	–	Cascading Style Sheets 3
GUI	–	Graphical User Interface
HTML5	–	Hyper Text Markup Language 5
JSON	–	JavaScript Object Notation
PNG	–	Portable Network Graphics
SQL	–	Structured Query Language
UML	–	Unified Modeling Language
VS	–	Visual Studio

Seznam obrázků

1 Úvod

V současné době tvoří informační technologie zásadní část našich životů a umožňují nám zjednodušit nebo automatizovat každodenní činnosti. Dnešní aplikace zvládají s přehledem celou řadu funkcí, jsou přehledné a jednoduché na používání. Informační systémy uchovávají statisíce datových záznamů a obslouží současně desítky tisíc uživatelů, tím ale narůstá i náročnost a komplexnost takovýchto systému. Systémy už nemohou být naprogramovány jedním člověkem nebo malou skupinou lidí bez řádného plánování. Systém jako takový je nejprve nutné navrhnout, analyzovat, posléze přizpůsobit požadavkům zákazníka a tyto požadavky zahrnout i ve výsledném řešení. Taktéž je důležité reagovat na potřebné zákaznické změny ve fázi vývoje. Zákazník obvykle nemá konkrétní představu, jak bude celý systém fungovat, tato představa se tvoří během fází vývoje. Pro standardizovaný vizualizovaný návrh a analýzu se používá jazyk UML. Jazyk UML je modelovací standardizovaný jazyk, který poskytuje nástroje pro specifikaci, vizualizaci, konstrukci a dokumentaci jednotlivých částí systému. Kromě popisu jednotlivých částí dává i možnost náhledu na komplexnost celého systému a pomáhá při stanovení ceny systému.

Cílem této bakalářské práce je implementace webové aplikace pro tvorbu třídních diagramů, která umožní generování entitní a databázové vrstvy. Práce se skládá z analýzy a návrhu, samotné implementace webové aplikace, uživatelské a programátorské dokumentace. Součástí je také krátký popis testování. Cílem práce je rovněž popis problematiky tvoření třídních diagramu a jejich převodů na kód.

Samotná webová aplikace bude spustitelná pomocí některého z běžných internetových prohlížečů a to bez jakékoliv nutnosti instalování dalších rozšiřujících technologií. Webová aplikace bude poskytovat veškeré potřebné nástroje pro vytváření a úpravu třídních diagramů. Současně je také kladen velký důraz na jednoduché a intuitivní ovládání. Aplikace také umožní generování zdrojových kódů. Součástí zdrojových kódů bude vygenerována entitní a databázová vrstva pro práci s databází v C# jazyce, stejně jako SQL script pro vytvoření samotné databáze. Mimo vygenerované kódy bude aplikace nabízet i grafický výstup namodelovaného třídního diagramu v podobě PNG obrázku.

V úvodu práce nastiňujeme problematiku vývoje aplikací a informačních systémů. Dále stručně popisujeme UML jazyk a věnujeme se podrobně popisu třídního diagramu, jeho možnosti zakreslování, generování kódů a způsobu generování relační databáze z třídního diagramu. Následuje analýza a návrh samotné aplikace, ve které bude možné modelovat třídní diagramy a generovat příslušné zdrojové kódy. Nedílnou součástí práce je uživatelská dokumentace, ve které je popsáno, jak se s danou aplikací pracuje, jak se používají její jednotlivé nástroje a jaké uživatelské rozhraní nabízí. Nezbytností je rovněž programátorská dokumentace, která zahrnuje použité technologie, popis jednotlivých metod a implementační problémy. Předposlední kapitola popisuje způsob testování vygenerovaných zdrojových kódů. V závěru práce jsou popsány splněné cíle celé bakalářské práce a vytvořené webové aplikace. Závěr také zahrnuje plány do budoucna i možné rozšíření aplikace.

2 Třídní diagram

V této kapitole si představíme a podrobně rozebereme třídní diagram. Specifikujeme jednotlivé vazby, možnosti generování kódu a způsob převodu na relační databázový model.

2.1 Unified Modeling Language

Jazyk UML je univerzální jazyk pro grafické modelování informačních systémů. Kombinuje existující postupy modelovacích technik a softwarového inženýrství [1]. Jedná se o důležitý nástroj, který umožňuje pomocí různých prostředků odhadnout náročnost a rozsáhlost informačního systému již ve fázi návrhu a vývoje. Stejným způsobem umožňuje provádět potřebné změny požadované ze strany zákazníka v průběhu vývoje a implementace. UML slouží jako dokumentace, čímž usnadňuje programátorům orientaci v příslušném systému a umožňuje rychlou orientaci v jeho jednotlivých částech. Jazyk UML je v současné době tvořen 18 diagramy, pomocí kterých se vizuálně popisuje informační systém. Každý z diagramů popisuje jinou část systému nebo stejnou část z jiného pohledu. Tyto diagramy jsou rozděleny do dvou hlavních skupin na diagramy struktury a diagramy chování. [2]

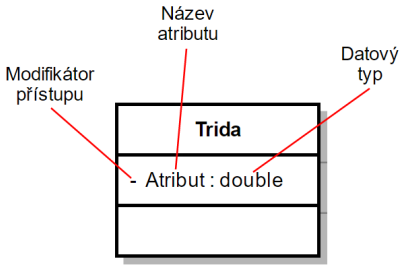
2.2 Třídní diagram

Třídní diagram patří do skupiny diagramů struktur. Graficky znázorňuje statický pohled na objektově orientovaný systém. Zachycuje množinu tříd, rozhraní a relací. Diagram je platformově závislý, konkrétní pro každý jazyk. Třídní diagram je abstrakcí objektového diagramu, nebo opačně, objektový diagram je instancí třídního diagramu [3].

Diagram tříd je diagram implementace. Obsahuje veškeré třídy, které by měl obsahovat napsaný program. Třídy obsahují potřebné atributy a metody. Tento diagram slouží jako jakýsi návod pro programátora, ten doplňuje potřebné chování a implementaci. Programátor tak neřeší zásadní otázky jak zkonstruovat daný systém. Tyto otázky jsou v rámci návrhu přenechány na zkušenější programátory a analytiky a to současně umožňuje přenechat implementaci méně zkušeným programátorům. Diagram také poskytuje náhled na celý systém z implementačního pohledu. To mnohdy odhalí problémové částí systému ještě před samotnou implementací. [4]

2.3 Atributy

Atributy představují typové vlastnosti objektu. Zapisovány jsou spolu s datovými typy v konkrétním implementačním jazyce. Jejich hlavním úkolem je uchovávat data. Atributy se zapisují do druhé části třídy. Na začátku každého atributu je modifikátor přístupu, následovaný názvem atributu. Modifikátory přístupu určují viditelnost atributů nebo metod v rámci celé aplikace. Existují čtyři modifikátory přístupu, privátní (private), veřejný (public), chráněný (protected) a modifikátor označující viditelnost v rámci celého balíku (package). Poslední částí atributu je datový typ. Název a datový typ jsou rozděleny dvojtečkou (Obrázek 1). [5]



Obrázek 1: Ukázka jednoduchého zápisu atributu

2.4 Metody

Metody reprezentují akce, které umí objekt (nebo v případě statické metody třída) vykonávat. Zapisují se standardně ve tvaru modifikátor přístupu, název, seznam parametrů, dvojtečka, návratová hodnota a v případě potřeby je specifikace metody ukončená vlastností ve složených závorkách. Mezi přiřaditelné vlastnosti patří redefines, query, ordered, unique a oper-constraint. Povinný je pouze název metody. Modifikátory přístupu se používají stejně jako u atributů. Násobnost je možné specifikovat u předávaných parametrů nebo v návratové hodnotě. Abstraktní metoda se značí úpravou fontu na italic, druhou variantou je použití rozšiřující vlastnosti „abstract“. [6]

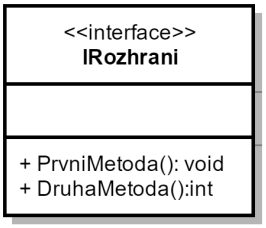
2.5 Třída

V třídních diagramech je třída tvořena názvem, atributy a metodami. Povinný je pouze název (Obrázek 2). Ten je zapisován vždy tučným písmem v první části samotné třídy, v případě abstraktní třídy je použita ještě navíc kurzíva. Nejjednodušší implementace třídy představuje implementaci pouze pomocí názvu.

Obrázek 2: Třída s atributy a metodami

2.6 Rozhraní

Rozhraní umožňuje přidat stejnou funkčnost rozdílným třídám. Rozhraní deklaruje funkce bez dané implementace, tu zajišťují příslušné třídy, které rozhraní implementují. V třídním diagramu se rozhraní zapisuje podobně jako třída. Abychom byli schopni oddělit třídu od rozhraní, používáme v zápisu takzvaný stereotyp, ten rozlišuje smysl elementu (Obrázek 3).



Obrázek 3: Rozhraní

2.7 Relace

V třídním diagramu relace představují způsob, jak zachytit komunikaci mezi jednotlivými elementy [7]. Aby bylo možné zachytit různé vztahy mezi třídami, je nutné specifikovat rozdílné relace:

- Závislost - Jedná se o nejslabší vazbu. Specifikuje vztah, kdy jeden nebo více elementů používá jiný element. Zakresluje se přerušovanou čárou s šipkou na konci ve směru vztahu.
- Asociace - Vyjadřuje základní vztah mezi elementy. Oba elementy mohou existovat nezávisle na sobě. Existuje hned několik druhů asociace, ty jsou rozlišovány hlavně podle násobnosti a řiditelnosti. Zakresluje se plnou čárou.
- Agregace - Je hodně podobná asociaci, popisuje vztah celek – část, kdy celek může existovat bez části. Škola reprezentuje celek, vidí studenty a současně studenti vidí školu. Pokud budou prázdniny a škola (celek) nebude mít žádné studenty (část), škola bude pořád existovat. Zakresluje se plnou čárou s nevyplněným kosočtvercem u celku.
- Kompozice - Stejně jako agregace, kompozice představuje vztah typu celek – část. Celek však nemůže existovat bez svých částí. Jídelníček představuje celek, ale bez svých položek nemůže existovat. Zakresluje se plnou čárou s vyplněným kosočtvercem u celku.
- Generalizace - Reprezentuje v třídním diagramu dědičnost. Dědičnost je jedna ze základních vlastností objektově orientovaného programování. Potomek dědí z předka, dědí se vlastnosti i chování. Značí se plnou čárou a nevyplněnou uzavřenou šipkou u předka.
- Realizace - Reprezentuje vztah mezi třídou a rozhraním, slouží ke znázornění implementace rozhraní. Značí se přerušovanou čárou s uzavřenou nevyplněnou šipkou u rozhraní.

U relací jako závislost, asociace, agregace a kompozice určujeme násobnost a řiditelnost. Řiditelnost označuje viditelnost mezi elementy. Násobnost určuje počet obsažených instancí datového typu, zapisuje se číselně ve formě rozsahu čísel. [8]

2.8 Generování zdrojových kódů z třídního diagramu

Navržený třídní diagram slouží jako šablona pro programátora a do jisté míry umožňuje generovat zdrojové kódy automaticky na základě údajů obsažených v příslušném diagramu. Automatické generování kódů značně usnadňuje i práci samotného programátora a díky vygenerované entitní vrstvě šetří čas, který je potřebný k programování. Současně také standardizuje kód, což zamezuje rozdílnému přístupu k implementaci alespoň v rámci entitní vrstvy. Zdrojový kód jde prakticky generovat ze všech částí třídního diagramu, třídy s jejich atributy a metodami můžeme jednoduše zaměnit za třídy s funkcemi a proměnnými, relace reprezentují reference na jiné objekty.

2.9 Mapování objektů na relační databázový model

Mapování objektů na relační databáze vždy ukrývá několik nástrah a problémů. Ty jsou způsobeny rozdílnými principy. Objektově orientovaný přístup, který využívají aplikace, se soustředí na koncepce, jako jsou dědičnost, zapouzdření, polymorfismus a další. Tento přístup je znám ze softwarového inženýrství a počítá s tím, že objekty budou vlastnit jak data, tak chování (metody). Na rozdíl od aplikací je databáze striktně určena pro ukládání dat. Tento základní rozdíl však způsobuje ne zrovna ideální kombinaci dvou paradigmat. Například mapování adresy, která je v aplikaci rozdělená do tří atributů (město, ulice, číslo popisné), je vhodnější do databáze zapsat jako jeden záznam. [9]

2.9.1 Mapování dědičnosti

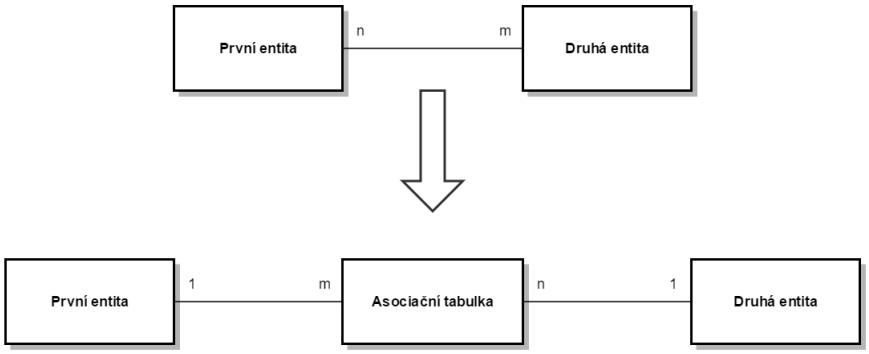
Mezi další problémy patří způsob řešení dědičnosti. Existuje hned několik způsobů, jak tento problém vyřešit, žádný z nich ale není ideálním řešením pro každý problém. Mezi základní řešení mapování dědičnosti patří:

- Jedna tabulka odpovídá celé hierarchii tříd.
- Jedna tabulka odpovídá jedné třídě. Tabulky potomků mají atributy předků.
- Jedna tabulka odpovídá jedné třídě. Tabulky potomků nemají atributy předků.

Žádná z těchto možností však není univerzální, proto je vždy nutné správně analyzovat požadavky na databázi a podle toho i přizpůsobit řešení. [10]

2.9.2 Mapování relací

V databázích jsou relace řešeny pomocí cizích klíčů, díky kterým se můžeme odkazovat u daného záznamu v první entitě na záznam nebo záznamy v druhé entitě. Tímto způsobem můžeme vyřešit vztahy 1:1 a 1:n. Vztah n:m je potřeba vyřešit asociační tabulkou, která obvykle obsahuje primární klíče obou entit. Reálně pak ze vztahu n:m vznikne vztah 1:m mezi první entitou a asociační tabulkou, a n:1 mezi asociační tabulkou a druhou entitou (Obrázek 4). [9]



Obrázek 4: Asociační tabulka

2.10 Srovnání aplikací pro vytváření třídních diagramu

Existuje nespočet aplikací umožňující vytváření a manipulaci s UML diagramy. Naneštěstí většina z nich přistupuje k modelování třídního diagramu jako k modelování ostatních UML diagramu, bez možnosti rozšířeného generování zdrojových kódů. My si některé z nich krátce představíme.

2.10.1 GenMyModel

Jeden z mála webových modelovacích nástrojů umožňující tvorbu třídních (a jiných) diagramů s možnostmi generování zdrojového kódu do Javy a s možností generování SQL scriptu pro vytvoření databáze. Aplikace má přehledné uživatelské rozhraní a ovládá se velice jednoduše. Freewarová verze stačí dostatečně na běžné projekty a poskytuje všechny výše uvedené možnosti. Pro spuštění je nutná registrace. Komerční verze pro společnosti poskytuje více nástrojů pro podporu vývoje a spolupráci v týmech.

2.10.2 Creately

Creately je multifunkční webový Flash editor pro vytváření diagramů. Podporuje několik grafických jazyků včetně UML. Přestože podporuje třídní diagram, tak neumožňuje generování kódů. Aplikace má chaoticky rozložené nástroje a některé i jednoduché operace je nutné složitě dohledávat. Všeobecně aplikace působí spíše jako editor různých tvarů než plnohodnotný UML

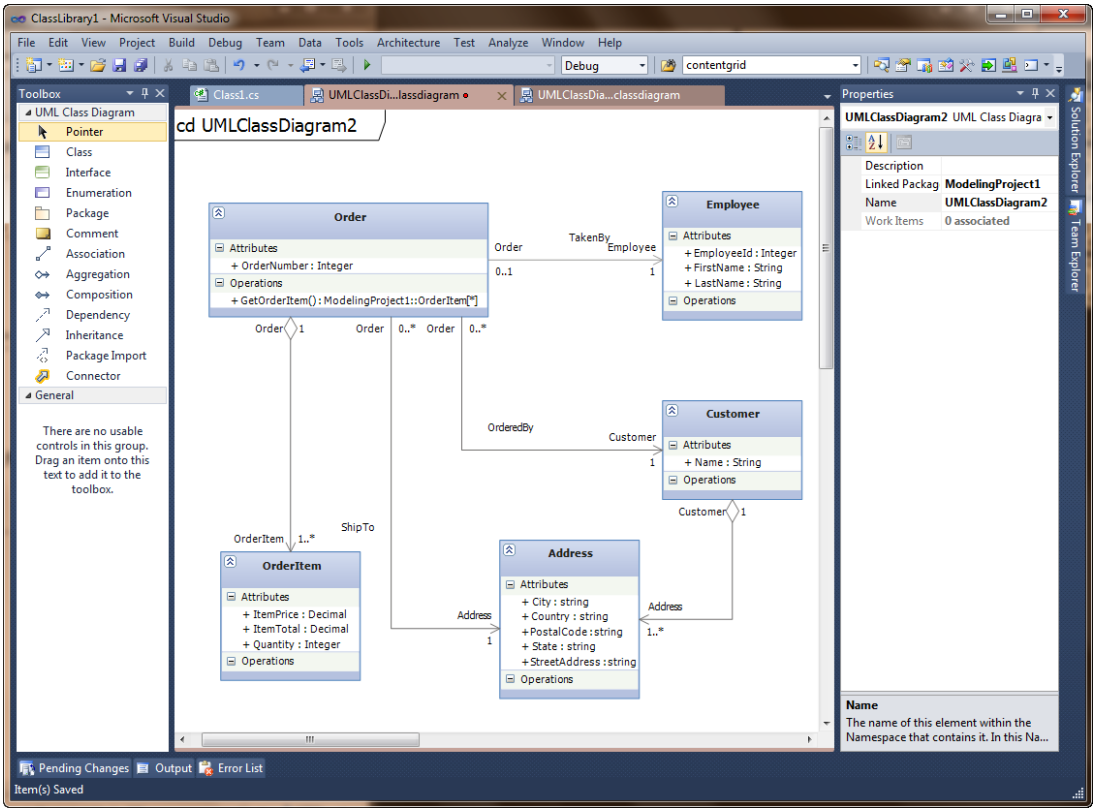
editor. Aplikaci jde spustit bez registrace, nicméně pro plnou podporu všech funkcí jako je uložení, export a podobně je nutná registrace a přihlášení. Komerční použití poskytuje daleko širší podporu, hlavně v možnostech exportu a práce více uživatelů na jednom projektu. Vytvořené diagramy působí vizuálně velice pěkně a není nutné je žádným způsobem dále upravovat.

2.10.3 Visual paradigma

Nejedná se o aplikaci jako takovou, ale o plugin pro Eclipse, který umožňuje modelovat 13 rozdílných UML diagramů včetně třídního diagramu. Řídí se UML standardem 2.0. Všeobecně se jedná o vydařený plugin, který poskytuje mnoho možností. Vytváření diagramů je přehledné a jednoduché. Umožňuje generovat kód do Javy, C# a Visual Basicu.

2.10.4 Visual Studio

Visual Studio (VS) samo o sobě nabízí nástroje, pomocí kterých můžeme vytvářet diagramy. Pomocí nástrojů Class View Designer a Class Diagram jde jednoduše namodelovat třídní diagram. Vytváření diagramů je velice rychlé a jednoduché. VS umožňuje vygenerovat kód přímo do používaného projektu. Také umí zpětně vytvořit diagram z již napsaného kódu, což značně pomáhá, pokud nenavrhujeme nový systém, ale chceme analyzovat nebo vylepšit již rozpracovaný systém a nemáme řádnou dokumentaci (Obrázek 5).



Obrázek 5: Třídní diagram ve VS

3 Analýza a návrh aplikace

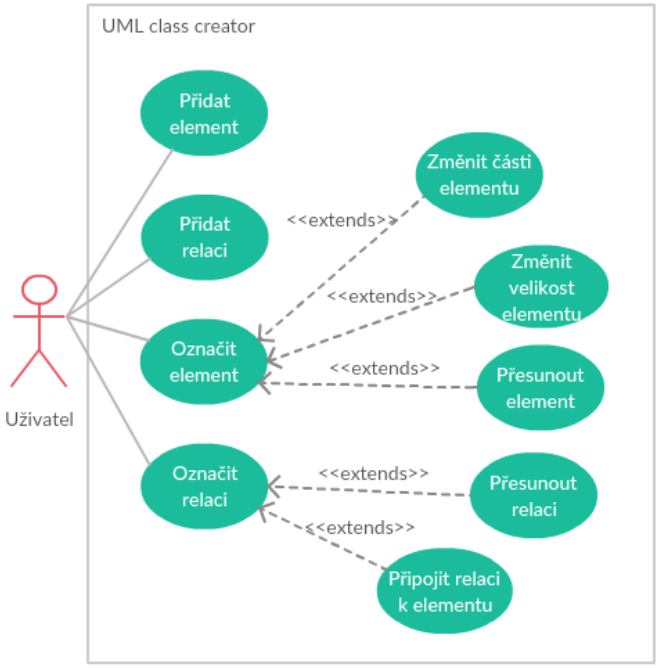
V této kapitole si specifikujeme požadavky na danou webovou aplikaci z pohledu uživatele, a popíšeme navržené řešení.

3.1 Specifikace zadání

Cílem práce je implementace webové aplikace. Aplikace bude přístupná pomocí webového prohlížeče a bude schopna sloužit jako plnohodnotný editor pro vytváření třídních diagramů. Dále aplikace umožní generování entitní vrstvy. Součástí vygenerovaných kódů bude i mapper, ve kterém budou naimplementovány základní operace pro práci s databázovou vrstvou. V poslední řadě bude aplikace generovat vrstvu pro práci s databází a SQL skript pro vytvoření databáze. Všechny výše uvedené kódy budou generovány v C# jazyce.

3.2 Specifikace požadavků

Základní specifikaci zadání jsme si již objasnili, nicméně aplikace jako taková se skládá z celé řady menších částí, které musí fungovat nezávisle na sobě nebo naopak, v kombinaci s jinými částmi. Tyto části potřebujeme pro základní funkčnost aplikace a jsou popsány v přiloženém diagramu (Obrázek 6).

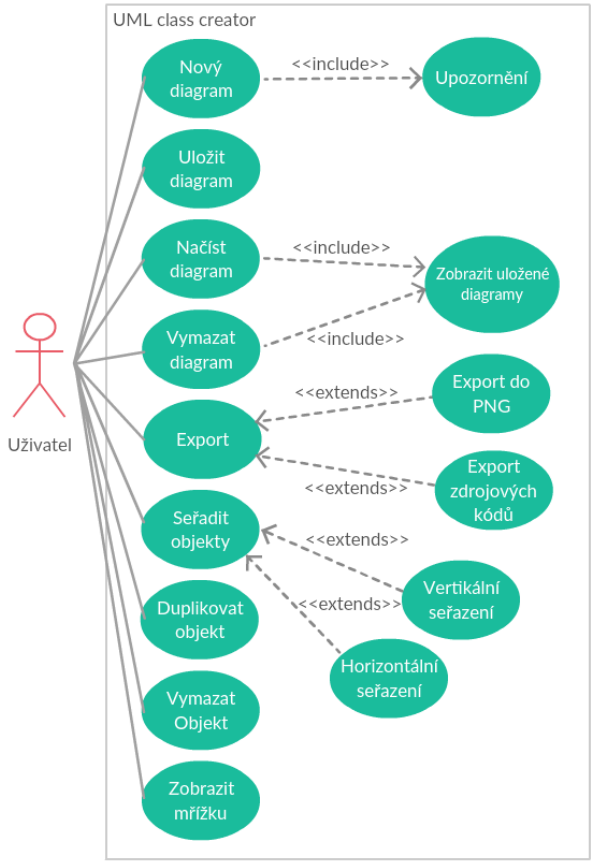


Obrázek 6: Diagram případu užití základní funkčnosti aplikace

Z přiloženého diagramu vidíme hned několik zásadních prvků, které budou důležité při vytváření aplikace. Uživatel bude mít možnost přidat element, tento případ užití představuje množinu operací, ve které můžeme přidat jeden z použitelných elementů třídního diagramu, například třídu, rozhraní, datatype, enumeration a podobně. Aplikace musí také umožnit manipulování s jednotlivými elementy. Elementy by mělo být možné přesouvat a měnit i jejich vnitřní strukturu a velikost. Stejně tak druhý případ užití (use case) znázorňuje přidání některé z relací. Relace musí být možné stejně jako elementy přesouvat. Relace jsou vytvářeny hlavně z důvodu znázornění vztahu mezi dvěma elementy, proto je nezbytně nutné, aby vytvořené relace bylo možné napojit na požadovaný element.

Všechny tyto základní požadavky už formují základní rozložení samotné aplikace. Ze zatím dostupných informací víme, že by aplikace měla být složena minimálně ze dvou grafických části. První část představuje vykreslovací prostor, do kterého bude uživatel umisťovat elementy a relace, a s nimi dále provádět příslušné operace. Druhá část označuje nabídku, ve které budou všechny druhy elementů a relací, které si následně bude moct uživatel vybrat a přidat je do vykreslovacího prostoru.

Aplikace má ale poskytovat mnohem širší spektrum možností. Ty jsou označeny ve druhém přiloženém diagramu (Obrázek 7).



Obrázek 7: Diagram případu užití pokročilé funkčnosti aplikace

Zatím jsme si specifikovali dvě části, které musí aplikace obsahovat (vykreslovací prostor a nabídku s elementy a relacemi). Třetí část bude obsahovat nabídku možností, kde půjde manipulovat s diagramem jako s celkem, současně zde budou obsaženy nástroje pro další manipulaci s objekty a samotným vykreslovacím plátnem.

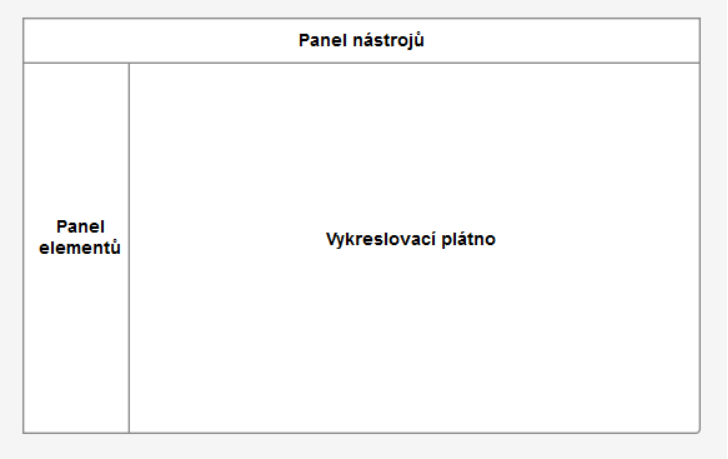
Uživatel by měl mít možnost začít zcela od začátku. Například pokud dokončí jeden diagram a bude chtít začít pracovat na novém, měla by existovat možnost začít nový projekt bez nutnosti aktualizace celé stránky. Proto bude tento nástroj zahrnut v možnostech aplikace. Dalšími možnostmi bude uložit, načíst nebo vymazat diagram. Způsoby ukládání budou popsány v programátorské dokumentaci (kapitola 5). Důležitou součástí bude i export diagramu, ten bude rozdělen do dvou hlavních částí. První částí bude export vizuální, který umožní převést diagram na obrázek, který si může uživatel stáhnout a ve druhé části bude zahrnut archiv s vygenerovanými zdrojovými kódy. Vzhledem k vizuálnímu exportu je nutné i graficky upravovat elementy. Ty bude možné horizontálně nebo vertikálně zarovnat. Pro lepší orientaci půjde na vykreslovacím plátně také zobrazit mřížka, která zvýší přehlednost jednotlivých elementů. Poslední část specifikace zahrnuje další manipulaci s objekty (elementy a relacemi) v diagramu. V rámci této manipulace půjde vybraný objekt odstranit nebo zkopírovat.

3.3 Návrh aplikace

Cílem návrhu je správně rozvrhnout architekturu samotné aplikace a zvolit správnou technologii, ve které bude aplikace vytvořena. Mezi vhodné webové technologie patří Adobe Flash nebo HTML5. Adobe Flash je technologie pro tvorbu interaktivních webových aplikací. Flash aplikace ke svému spuštění potřebují Flash player, který není obsažen ve všech prohlížečích a je potřeba jej zvlášť doinstalovat. Další nevýhodou je stále více diskutovaná bezpečnost Flash playeru, která může některé uživatele odradit. Z tohoto důvodu poslední dobou vývoj Flashových aplikací upadá a razantně se zmenšuje i podíl webů, které tyto aplikace využívají. Flashové aplikace jsou postupně vytlačovány technologií HTML5. Vzhledem k nejasné budoucnosti Flash aplikací bude naše aplikace vytvořena primarně v kombinaci HTML5, CSS3 a JavaScriptu.

3.3.1 Návrh uživatelského rozhraní

Ze specifikace požadavků víme, že potřebujeme rozdělit aplikaci do tří (grafických) části. Díky specifikaci požadavků můžeme navrhnout uživatelské rozhraní. Na wireframu můžeme vidět tři části aplikace rozvržené do patřičných celků (Obrázek 8). V aplikaci bude potřeba specifikovat ještě další dvě části. Jedna bude sloužit pro výstražná upozornění a druhá jako vyskakovací panel obsahující kontextové informace.



Obrázek 8: Wireframe uživatelského rozhraní

3.3.2 Rozvržení prvků a použité knihovny

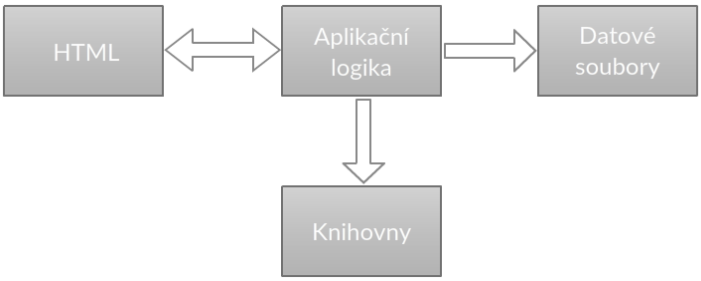
V závislosti na tom, že aplikace obsahuje značné množství interaktivních prvků, bude nutné tomu přizpůsobit i příslušné řešení. Hlavní část aplikace, tedy jak ji máme navrženou ve wireframu, bude zobrazována v prvku jménem canvas, který je součástí HTML5. O aplikační logiku se postará JavaScript. Vyskakovací panel a další prvky budou zobrazovány pomocí kombinace HTML5 a CSS3. Vyskakovací panel bude zobrazovat rozdílné informace, ty budou tematicky vždy odpovídat požadované funkcionalitě. V případě, že uživatel zvolí možnost načíst diagram, budou v panelu zobrazeny všechny uložené diagramy.

Nezbytnou součástí aplikace jsou i JavaScriptové knihovny třetích stran. Konkrétně budou využity 3 knihovny:

- REIMG – Knihovna, která zpracuje canvas a vyexportuje ho jako png obrázek. Díky tomu můžeme exportovat vytvořené diagramy.
- JsZip – Knihovna, která umí vytvářet zip archivy na klientské straně v prohlížeči. Vzhledem k bezpečnosti, kterou prohlížeče poskytují, není reálně možné pracovat se soubory na klientském zařízení. Nicméně tato knihovna je schopná vytvořit tyto soubory v paměti prohlížeče, i ta je však omezená a příliš velké zip soubory nezvládá. Naše aplikace bude vytvářet textově založené soubory velikostně maximálně několik desítek kilobytu a proto je toto řešení dostačující.
- FileSaver – Malá knihovna, která pracuje s HTML5 File API a umožňuje stáhnout soubor vytvořený na klientské straně.

3.3.3 Struktura aplikace

Z implementačního pohledu se struktura aplikace dělí do 4 částí.



Obrázek 9: Struktura webové aplikace

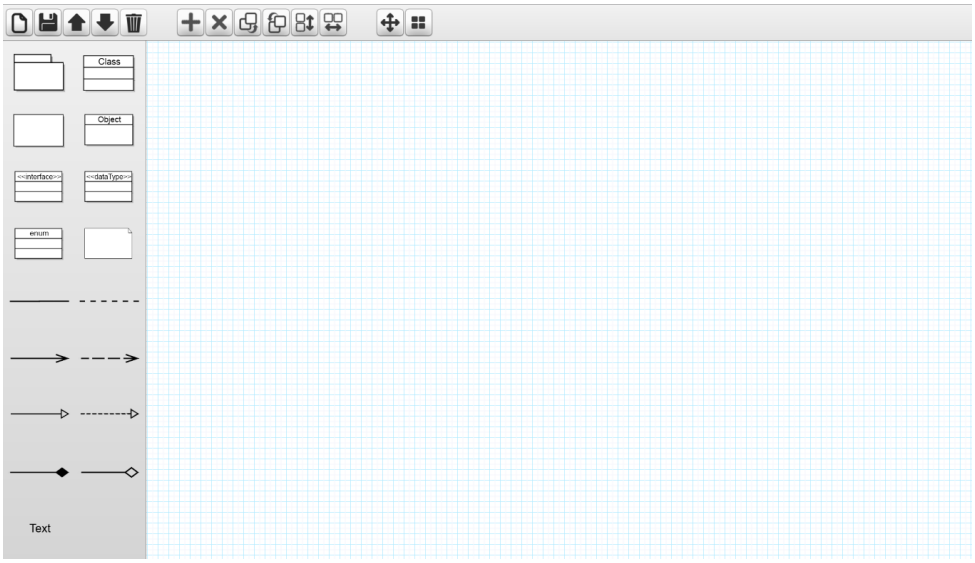
První část reprezentuje HTML soubor, který je základem webové aplikace. Ten bude svůj obsah měnit na základě uživatelské interakce. Druhá část označuje JavaScriptový soubor, který obsahuje aplikační logiku. V tomto souboru jsou řešeny veškeré operace spojené s fungováním aplikace. Aplikační logika je podrobně popsána v páté kapitole. Třetí část je tvořena malými JavaScriptovými soubory, které uchovávají datové typy a další informace o konkrétních programovacích jazycích a databázích, což umožňuje jednoduché rozšíření aplikace, která může být v budoucnu žádoucí. Poslední část označuje JavaScriptové knihovny třetích stran (Obrázek 9).

4 Uživatelská dokumentace

V této kapitole si popíšeme, jak naši aplikaci používat a jak jednotlivě ovládat její dílčí prvky. Všechny příklady budou vysvětleny a k složitějším procesům budou přidány grafické ukázky z aplikace.

4.1 První spuštění a orientace v aplikaci

Webová aplikace je spustitelná pomocí webového prohlížeče. Mezi podporované a testované prohlížeče patří Google Chrome, Mozilla Firefox, Microsoft Edge a Internet Explorer 11. Microsoft Edge a Internet Explorer 11 požadují pro spuštění povolení blokovaného obsahu. Webové prohlížeče Safari a Opera nebyly testovány, nicméně podle technologické dokumentace by měly být schopny aplikaci zvládnout. Aplikace může být nekompatibilní s nižší verzí Internet Explorer. Po spuštění aplikace je uživatel přímo uveden do samotného editoru třídních diagramů.



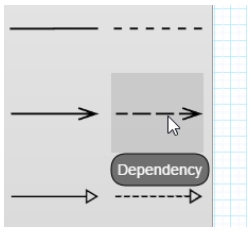
Obrázek 10: Ukázka webové aplikace

Celá aplikace je rozdělena do tří hlavních sektorů. Horní panel slouží pro celkové ovládání aplikace a dále poskytuje speciální nástroje pro manipulaci se všemi prvky v aplikaci. Panel umístěný vlevo slouží pro výběr jednotlivých prvků, které reprezentují třídní diagram. Největší grafická část je zabraná pracovní plochou, kde je možné přidávat jednotlivé prvky a také s nimi manipulovat (Obrázek 10).

4.2 Levý panel a přidání prvků

Levý panel poskytuje uživateli rozsáhlou nabídku prvků, které může následně přidat na pracovní plochu. Mezi umístitelné elementy patří balíček, třída, zjednodušená třída, objekt, rozhraní, datatype, enumeration a poznámka. Mezi umístitelné vztahy patří asociace, přerušovaná čára

spojující element a poznámku, přímá asociace, závislost, generalizace, implementace, agregace, kompozice a text. Při najetí kurzorem myši na jednotlivé prvky se po krátké chvíli zobrazí i pomocný text, který reprezentuje název elementu nebo vztahu, ten následně může pomoct méně zkušeným uživatelům v identifikaci prvku (Obrázek 11).

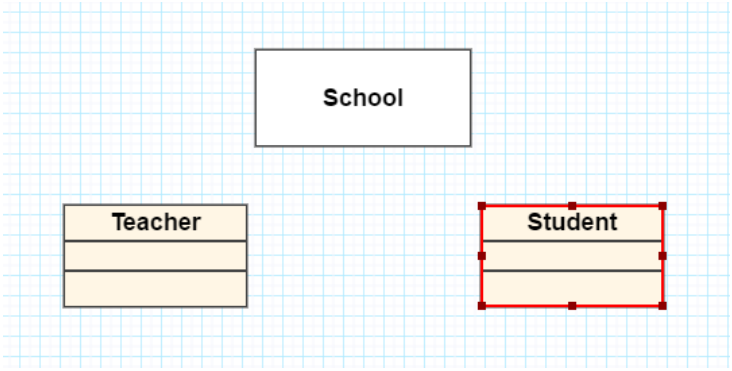


Obrázek 11: Příklad zobrazení pomocného textu

Přidání jednotlivých prvků je velice jednoduché a intuitivní. Nejdříve je nutné vybrat prvek, který chce uživatel přidat. Následně lze tento prvek jednoduše přetáhnout na pracovní plochu, kde se po puštění ikony objeví upravitelný prvek.

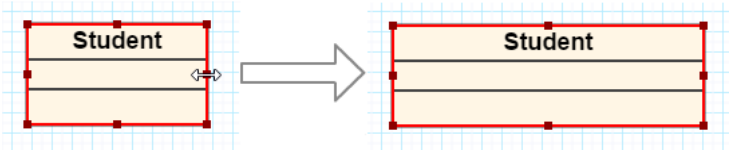
4.3 Výběr elementu a jeho úprava

Před jakoukoliv úpravou elementu je nejprve nutné takový element označit. Označení provádíme levým kliknutím myši na potřebný element. Označený element je následně ohraničen červenou barvou (Obrázek 12).



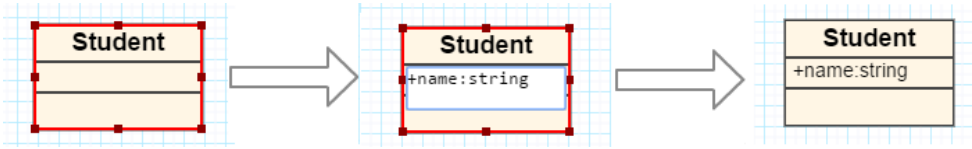
Obrázek 12: Rozdíl mezi označeným a neoznačeným elementem

U označeného elementu jsou zobrazeny i body, pomocí kterých jde element jednoduše zmenšit nebo zvětšit. Pokud uživatel ukáže kurzorem na jeden z těchto bodů, změní se styl kurzoru a následovným tažením se změní i velikost elementu v závislosti na vybrané straně. Minimální velikost elementu je přizpůsobena textu (Obrázek 13). Označený element jde také přesunout. Přesunutí se provádí jednoduchým uchopením a následným puštěním na jiné pozici.



Obrázek 13: Změna velikosti elementu

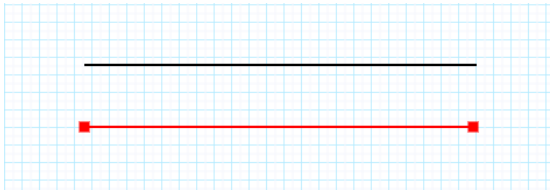
Změna jednotlivých textových částí elementu se také provádí na označeném elementu. Nad takto označený element přesuneme kurzor, následným dvojklikem levým tlačítkem myši se přesuneme do editačního režimu. Každý element má hned několik částí, které jde upravit. V případě třídy se jedná o 4 části. Mezi ně patří jméno, atributy, metody a násobnost. Každá část se upravuje zvlášť. Při úpravě je nutné dodržet předepsaný formát. Pokud skončíme s úpravou, editační režim zrušíme výběrem jiného nebo stejného elementu. Editační režim jde také zrušit kliknutím mimo jakýkoliv element (Obrázek 14).



Obrázek 14: Editační režim a změna atributu

4.4 Výběr relace a její úprava

Stejně jako u elementu jde jakoukoliv úpravu provádět pouze u označené relace. Označení se provádí levým kliknutím myši na požadovanou relaci. Označená relace je od neoznačené rozlišena červenou barvou. U každé označené relace se objeví dva pomocné body, díky kterým jde relace přesunout (Obrázek 15).

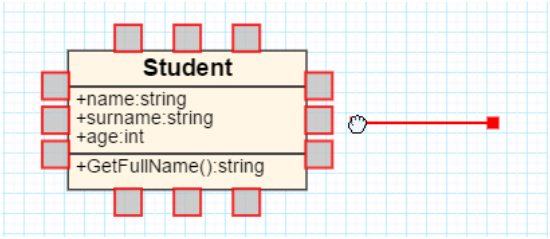


Obrázek 15: Neoznačená a označená relace

Pokud najedeme kurzorem nad pomocný bod, změní se i styl kurzoru. Pomocný bod jde následně levým tlačítkem myši chytit a přetáhnout na jinou pozici. Současně jde přesouvat pouze jeden bod (Obrázek 16).

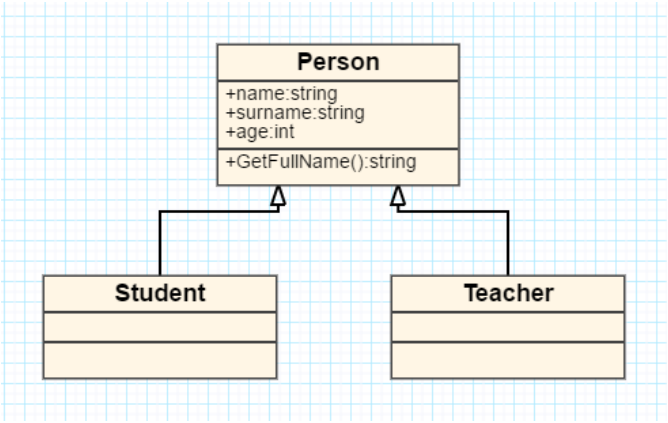
Obrázek 16: Přesun relace

Relace jde tímto způsobem také připojit k danému elementu. Při tažení pomocného bodu se u všech přítomných elementů zobrazí přípojné oblasti. V případě, že zde jeden z bodů přesuneme, je relace částečně připojena. Stejným způsobem jde připojit i druhá strana relace (Obrázek 17).



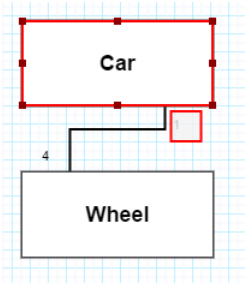
Obrázek 17: Přípojení relace k elementu pomocí přípojných oblasti

V případě, že připojíme obě strany relace a následně zrušíme označení, je relace překreslena do výsledné formy, kde jsou obě strany spojené pomocí několika pravoúhlých úseček. Připojení relace k jinému elementu nebo jeho části vykonáme opětovným označením relace a přetažením pomocného posuvného bodu (Obrázek 18).



Obrázek 18: Příklad vykreslení relací

Po připojení relace k elementu se po opětovném označení daného elementu objeví nové okénko. Do tohoto okénka jde stejným způsobem jako při předchozí textové editaci zapsat násobnost u jednotlivých relací (Obrázek 19).



Obrázek 19: Příklad násobnosti

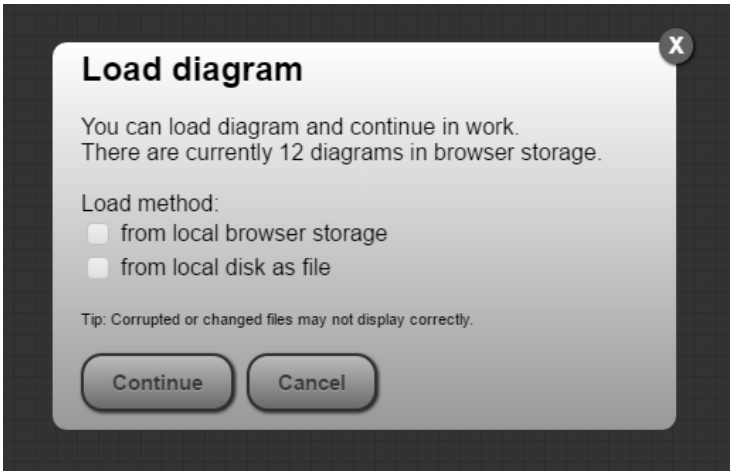
4.5 Horní panel a jeho funkce

Horní panel je v aplikaci rozdělen do tří pomyslných částí, které jsou odděleny pomyslnými mezerami.

4.5.1 První část horního panelu

První část reprezentuje akce týkající se diagramu. Umožňují jednotlivé manipulování s diagramy jako takovými a poskytují celou řadu užitečných možností. Mezi možnosti patří:

- Nový diagram - Umožňuje začít zcela nový projekt bez nutnosti aktualizace celé stránky.
- Uložit diagram - Umožňuje uložit diagram a pokračovat v práci později. Uložení je možné provést dvěma způsoby. První způsob ukládá data reprezentující diagram do paměti prohlížeče. Tato data přetrvávají i po vypnutí prohlížeče a počítače a je možné je kdykoliv načíst zpět. Nicméně pokud uživatel tuto paměť vymaže, nenávratně tak přijde o svá data. Stejně tak se tato paměť vztahuje pouze na současně používaný prohlížeč. Není tedy možné vytvořit diagram, uložit ho do paměti prohlížeče a následně v jiném prohlížeči tento diagram načíst. Z tohoto důvodu vznikl druhý, více přenosný způsob ukládání. Ten vytváří datový soubor, který si uživatel stáhne a následně uloží na svém disku.
- Načíst diagram - Umožňuje načíst diagram. Načítat lze z paměti prohlížeče nebo z datového souboru. Při vybrání této možnosti aplikace také ukazuje počet a výpis všech uložených diagramů z paměti prohlížeče (Obrázek 20).



Obrázek 20: Vyskakující okno s možností načtení diagramu

- Exportovat diagram - Umožňuje exportovat diagram. První možností je export grafický, kdy aplikace exportuje diagram do PNG obrázku tak, jak je nakreslen. Druhá možnost umožní vytvoření odpovídajících zdrojových kódů do C# jazyka. Aplikace je schopná vygenerovat také entitní vrstvu a SQL script pro vytvoření databáze. Součástí je také

mapper, servisní a databázová vrstva. Mapper obsahuje základní dotazy pro jednotlivé třídy. Poslední volba nabízí vygenerování čistě zdrojových kódů nebo vytvoření projektu, který je kompatibilní s Visual Studiem.

- Smazat diagram - Umožňuje vymazat zvolený diagram z paměti počítače.

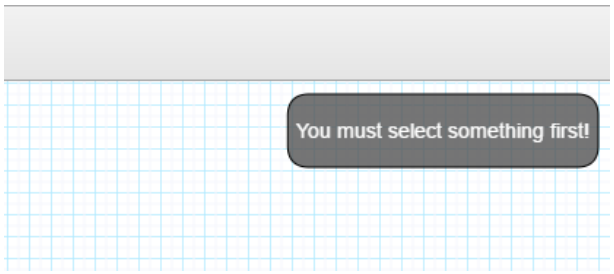
4.5.2 Druhá část horního panelu

Druhá část reprezentuje operace, které se dají provádět přímo s označenými elementy. Mezi možnosti druhé části patří:

- Duplikovat prvek - Po označení daného elementu nebo relace a vybrání této možnosti je prvek zkopírován.
- Vymazat prvek - Po označení elementu nebo relace je prvek smazán.
- Přesunout do popředí - Elementy umístěné na sebe se vzájemně překrývají. Tato možnost umožňuje přesunout označený element do popředí, což může být užitečné například při vytváření složitějších diagramů, kde využíváme vnořené balíčky.
- Přesunout do pozadí - Umožňuje přesunout vybraný element do pozadí.
- Vertikální synchronizace - Umožňuje vertikálně zarovnat dva elementy.
- Horizontální synchronizace - Umožňuje horizontálně zarovnat dva elementy.

4.5.3 Informační panel

Informační panel není ovládán uživatelem a slouží pouze jako informační nástroj. V panelu se zobrazují různá upozornění a případné informace pro uživatele. Také jsou zde zobrazena chybová upozornění, která upozorňují uživatele na chybné používání některých funkci nebo na nekompatibilitu používaných technologií se současně používaným prohlížečem. Zobrazuje se v pravém horním rohu pracovní plochy (Obrázek 21).



Obrázek 21: Chybová hláška zobrazena při kopírování elementu

5 Programátorská dokumentace

Implementace aplikace byla rozdělena do tří částí. V návrhové části jsme specifikovali všechny potřebné požadavky, ty pak bylo nutné v rámci implementace realizovat. V první části bylo potřeba vytvořit grafické podklady pro uživatelské rozhraní. Vytvořeny byly veškeré ikony a obrázky použitelné dle návrhu GUI. Ve druhé části se realizovala samotná implementace aplikace s využitím již vytvořených obrázků a návrhu GUI. Implementována byla i samotná logika aplikace. Třetí část probíhala současně s druhou částí a označuje testování aplikace. Testování bylo nesmírně důležité, zvláště z důvodu použití JavaScriptu, který je překládaný až za běhu samotné aplikace, což značně ztížilo debugování hlavně v posledních fázích vývoje.

5.1 Zvolené nástroje pro vývoj aplikace

Aplikace byla tvořena pomocí celé řady nástrojů a technologií. Grafická část aplikace byla vytvořena programem Adobe Photoshop CS6. Implementace probíhala v programu WebStorm 11 od společnosti JetBrains, ten kromě značné syntaktické kontroly nabízel i nástroje podporující debugování. V rámci testování byly použity webové prohlížeče Google Chrome, Mozilla Firefox, Microsoft Edge a Internet Explorer 11. K ověření správnosti vygenerovaných zdrojových kódů byl použit nástroj Visual Studio 2015 od společnosti Microsoft.

5.2 Použitá řešení aplikační logiky

Aplikační logika byla implementována podle návrhu v kapitole 3.3. Současně se kladl velký důraz na psaní komentovaného strukturovaného kódu, který se řídí podle objektově orientovaných pravidel.

Základem aplikace se stal již vytvořený zdrojový kód pro manipulaci s objekty na HTML5 canvasu, který nebyl součástí návrhu. Tento vytvořený kód byl součástí návodu, který popisoval možnosti přesouvání objektu na canvasu [11]. Prakticky tento kód umožňoval vytváření obdélníků na canvasu a jejich následný přesun a změnu jejich velikost. Tento přibližně 300 řádkový JavaScriptový soubor byl následně použit jako základ pro naši webovou aplikaci. V průběhu implementace naši webové aplikace byl tento zdrojový kód přepracován tak, aby vyhovoval potřebným požadavkům aplikace. Původní neupravený a převzatý kód odpovídá přibližně 4% délce finálního naimplementovaného kódu.

Oproti navrženému řešení byla provedena ještě malá úprava v rámci generování PNG obrázku. Pro generování PNG obrázku nebyla nakonec použita knihovna reimg z důvodu nekompatibility této knihovny s některými prohlížeči. K vyřešení tohoto problému bylo vytvořeno hned několik funkcí. Tento problém je detailněji popsán v kapitole 5.2.8.

5.2.1 Inicializační funkce a start aplikace

Jako první byly vytvořeny globální proměnné, které udržují základní hodnoty využívané celou řadou funkcí. Následně byla vytvořena funkce init, ta slouží jako inicializační funkce a je tedy spuštěna pouze při startu aplikace. V této funkci jsou nastaveny výchozí hodnoty nebo vytvořeny nové objekty při spuštění aplikace, nastavují se zde hodnoty jako výška a šířka uživatelské obrazovky, uživatelský prohlížeč, reference na HTML elementy a podobně. Dále jsou zde nastaveny funkce, které se mají vykonat, pokud nastane událost jako kliknutí, pohyb myši a dvojklik. Také je zde nastaven časovač, který v určitém intervalu volá funkci mainDraw. Tu si popíšeme v kapitole 5.2.2. V pokročilé části implementace zde přibylo vytvoření dalších objektů, které reprezentují jednotlivá tlačítka a vytvoření informačního panelu.

5.2.2 Funkce mainDraw a vykreslovaní

Funkce mainDraw je zodpovědná za volání všech vykreslovacích funkcí. Na začátku této funkce je volána jiná funkce na kontrolu velikosti okna, ve kterém je aplikace spuštěna. Poté jsou zde volány další funkce, které postupně vykreslují části viditelné na canvasu. Nejdříve jsou v cyklu vykresleny všechny objekty, reprezentující naše třídy, rozhraní a podobně a poté jsou stejným způsobem vykresleny relace. Každý z těchto objektů je zodpovědný za své vlastní vykreslování na canvas. Nakonec jsou vykresleny jednotlivé panely s tlačítky a informační panel. Aby byla zajištěna plná plynulost aplikace, je tato funkce volána intervalově v rozmezí patnácti milisekund. Při každém spuštění této funkce také probíhá kontrola, zda byla vykonána nějaká změna, kterou je potřeba promítnout na canvas. Pokud žádná změna nenastala, není třeba ani provádět nové vykreslování, což razantně snižuje potřebný výkon k běhu aplikace.

5.2.3 Objekt typu Button

Každé tlačítko je v aplikaci reprezentované objektem typu Button. Všechny tyto objekty jsou vytvořeny ve funkci init a při vytváření uloženy do globálního pole. Objekt typu Button vlastní hned několik proměnných. V proměnných je udržováno jméno, které je zobrazeno při najetí na tlačítko. Dále jsou zde uchovávány souřadnice a rozměry, které vymezují, kde bude tlačítko vykresleno. Objekt obsahuje také dvě proměnné typu Image, ty určují, jaký obrázek bude vykreslen. Zbytek proměnných umožňuje zobrazení hover efektu. Objekt typu Button má přiřazeny tři funkce, jsou to:

- drawButton - Funkce jednoduše vykresluje objekt na canvas, který je zde uveden jako parametr.
- checkPosition - Funkce má za úkol porovnat současnou pozici kurzoru uživatele a souřadnice tlačítka. Pokud se ve funkci zjistí, že kurzor ukazuje na tlačítko tak je změněna proměnná type, která přísluší tomuto objektu. Tím se při dalším překreslení změní vykreslovaný obrázek, což vykoná hover efekt. Současně je také pomocí této funkce zobrazen

popis, který identifikuje název tlačítka a je zobrazen pod tlačítkem v černém zaobleném rámečku. Při zobrazení popisu je použit efekt postupného zobrazení, kdy je během krátkého časového úseku zvyšována průhlednost vykresleného popisu a zaobleného rámečku (Obrázek 22).

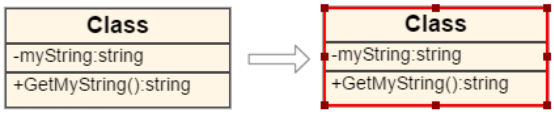
Obrázek 22: Tlačítko a hover efekt

- addBox - Funkce, která umožňuje přidat konkrétní elementy nebo relace.

5.2.4 Objekt typu Box

Všechny elementy použitelné v třídním diagramu (balíček, třída, objekt, rozhraní, datatype, enum, poznámka a text) jsou reprezentovány objektem typu Box. Ten v proměnných udržuje své souřadnice, současné rozměry, minimální rozměry a barvu výplně. Dále jsou zde čtyři pole, tři z nich udržují textové hodnoty, které reprezentují proměnné a metody přímo v diagramu. Poslední pole udržuje reference na všechny relace připojené k elementu (na objekty typu Connection). Objekt typu Box vlastní pět funkcí, mezi ně patří:

- draw - Má za úkol provést veškerá vykreslování. Vykreslován je obdélník reprezentující příslušný element. Každý rozdílný element má mírně upravenou barvu výplně, což je umožňuje lépe odlišit. Vykreslován je i příslušný text v rámci elementu, který odpovídá potřebným atributům a metodám. Samozřejmostí je i vykreslování názvu, stereotypu a násobností. Součástí této funkce je i detekce, která kontroluje, zda je tento objekt vybrán uživatelem. Pokud ano, je obtažen tmavě červenou barvou a současně jsou vykresleny menší čtverečky umístěné v každém rohu a uprostřed každé stěny (Obrázek 23). Celé vykreslování je prováděno na canvas, který je předán jako parametr.



Obrázek 23: Vykreslení elementu se zobrazením rozdílu při vybrání

- drawBoxes - Slouží pro vykreslování pomocných čtverců, do kterých je možné připojit relace. Tato funkce je zodpovědná pouze za vykreslování, logika připojení je v následující funkci checkBoxesPos. Pomocné čtverce jsou vykresleny pouze v případě, že je vybrána

některá z relací a uživatel drží některý z bodů relace (Obrázek 17). Vykreslování je rovněž jako u předchozího případu prováděno na canvas, který je předáván parametrem.

- checkBoxesPos - Funkce provádí logiku připojení relace k elementu. Tato funkce je spouštěna v případě, že jsme přesouvali některý z bodů relace a někam jsme ho umístili. Následně jsou postupně ověřeny všechny možné kombinace, kde mohl uživatel připojit relaci. Pokud zjistíme, že uživatel připojil, tedy umístil vybraný bod relace na jeden z pomocných čtverců elementu, vytvoříme vzájemné reference a nastavíme, kde byla daná relace připojena a ukončíme funkci. Funkce je bezparametrová.

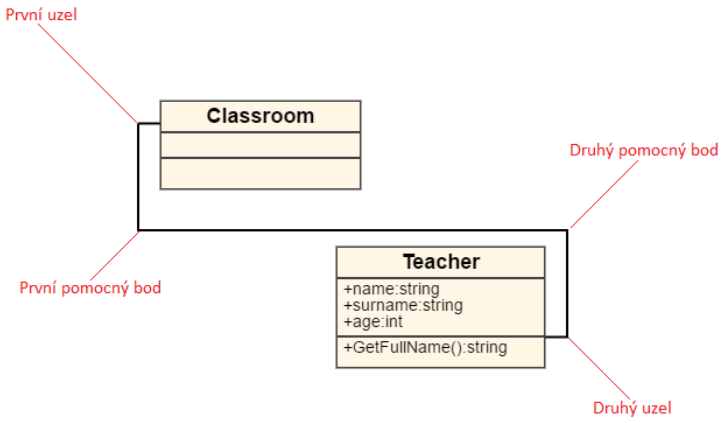
- calculateSize - Funkce kontroluje a nastavuje minimální velikost, jakou element může mít. Vzhledem k tomu, že v elementech figuruje text, je nutné nastavit minimální velikost, jakou element může mít. To vylučuje možnost přesahujícího textu přes hrany elementu. Ve funkci je nejdříve uložena původní minimální velikost elementu. Následně je spočítána nová velikost. Tyto dvě hodnoty jsou poté porovnány, pokud jsou hodnoty stejné, není nutné provádět změny. Pokud jsou hodnoty rozdílné, je třeba nastavit minimální velikost na nově vypočtenou hodnotu a provést změnu velikosti. Parametrem funkce je canvas, ten je potřebný k vypočtení rozměrů textu. Funkce je volaná pouze v případě změny některého z textů v daném elementu.

- resize - Funkce provádí změnu velikosti elementu. Je volána z funkce calculateSize. Rozměry jsou nastaveny na minimální možnou velikost elementu.

5.2.5 Objekt typu Connection

Veškeré relace jsou reprezentovány objektem typu Connection. Ten obsahuje dva objekty typu ConnectionNode, proměnnou type a kontrolní hodnoty. Všechny Connection objekty jsou přidány do globálního pole. ConnectionNode reprezentuje jeden uzel (bod), který uchovává potřebné informace o své poloze, hodnotu násobnosti relace, a referenci na připojený element (objekt typu box). Další dvě proměnné jsou pomocné a používají se při vykreslování. Objekt typu Connection má přiřazeny dvě funkce, draw a calculateBs:

- draw - Funkce vykresluje veškeré potřebné hodnoty na canvas, který získáváme pomocí parametru. V závislosti na hodnotě proměnné type, je u prvního uzlu vykreslována také speciální značka reprezentující konkrétní typ relace (agregace, kompozice a podobně). Stejně jako u elementů je vybraná relace označena červenou barvou. Pokud je relace vybrána, vykreslí se také dva čtverce na obou koncích relace, díky kterým je pak možné s relací pohybovat. Přesun relace řeší jiná funkce. Tato funkce je značné rozsáhlá, zvláště kvůli propracovanějšímu způsobu vykreslování. Dvě propojené relace jsou vždy spojené pomocí pravoúhlých úseček. Při pravoúhlém vykreslování se používají pomocné proměnné. Ty označují vnitřní rohové body (Obrázek 24).



Obrázek 24: Pomocné body u relací

- calculateBs - Funkce má za úkol vypočítat souřadnice pomocných bodů a je volána vždy při vykreslování relace.

5.2.6 Úprava jednotlivých částí elementu

Všechny části elementů by měly být jednoduše upravitelné. Aby byla úprava jednoduchá a současně efektivní, bylo nutné navrhnout důmyslnější mechanismus pro úpravu textu. Canvas v HTML5 nepodporuje žádnou úpravu textu, slouží pouze k vykreslování. JavaScriptové knihovny řešící problém úpravy textu na canvasu jsou příliš rozsáhlé a umožňují nespočet zbytečných funkcí, které bychom nevyužili. Proto bylo navrženo řešení používající HTML. Řešení je popsáno v kapitole 5.3.1.

5.2.7 Generování kódu

Celé generování je prováděno postupných spojováním předpřipravených hodnot. Hodnoty jsou uložené v proměnných a jsou součástí nezávislého JavaScriptovém souboru. To umožňuje jednoduchou editaci v případném budoucím rozšíření. Generování samotného kódu obstarává několik funkcí:

- GenerateFiles – Funkce, která výhradně pracuje s knihovnou JsZip. Má jeden parametr, ve kterém je předán typ generování. Tato funkce vytváří celou strukturu zip souboru. V případě, že uživatel zvolí generování čistých zdrojových kódu, je vytvořen adresář a následně zavolána funkce allClassesWithDB nebo AllClasses. Tyto funkce vrací pole objektů, které obsahují vygenerovány zdrojový kód a název dané třídy nebo rozhraní. V cyklu jsou pak všechny tyto řetězce vyexportovány do cs souborů, a vloženy do již vytvořeného adresáře. Ten je pak zabalen do zip archívu a nabídnut uživateli ke stažení. Druhá možnost nastává, pokud uživatel zvolí vyexportování projektu do VS. Tehdy je vytvořeno hned několik adresářů reprezentující strukturu projektu VS.

- allClasses – Tato funkce se stará o celé generování zdrojových kódu. V cyklu postupně procházíme všechny vytvořené objekty typu Box. Nejdříve vygenerujeme již předpřipravenou hlavičku třídy, poté získáme z našeho Box objektu název a ten přidáme do současně generovaného kódu. Poté stejným způsobem procházíme atributy a metody. Na konci je celá jedna třída vygenerována do jednoho řetězce, který je následně uložen do pomocného objektu. Pomocné objekty dále seskupujeme do pole, které pak vracíme na konci funkce. Celé generování však zahrnuje řadu podmínek, na základě kterých je rozhodnuto, jaká část kódu má být vygenerována. Kontrolují se například datové typy. Před samotným generováním atributů a metod se ještě kontrolují naše relace, které je nutné zahrnout ve vygenerovaných zdrojových kódech. Vygenerovaná třída pak obsahuje všechny proměnné a funkce. Současně také obsahuje instanci nebo instance objektů, které odpovídají namodelovaným relacím. Použití dědičnosti a implementace rozhraní se taktéž projeví ve vygenerovaných kódech.

- allClassesWithDB – Tato funkce vytváří pro každou třídu v diagramu tři třídy v zip archivu. První třída reprezentuje entitní vrstvu. Všechny atributy jsou zde vytvořeny jako vlastnosti (properties). V případě veřejného atributu je vytvořena vlastnost s defaultním gettrem a settrem. Pokud je atribut privátní, je vlastnost vytvořena jako veřejná a k tomu je navíc vytvořena privátní proměnná, včetně potřebného nastavení gettru a settru. Druhou vytvářenou třídou je třída reprezentující mapper. Zde jsou vytvořeny všechny funkce, které byly namodelovány v aplikaci. Také jsou zde naimplementovány základní operace s databází. Mezi tyto základní operace patří výběr záznamu podle ID, výběr všech záznamů, vložení záznamu, úprava a smazání záznamu. Poslední vygenerovanou třídou je servisní třída. Ta umožňuje volat metody mapperu. Ke každému vygenerovanému projektu je také přidána jedna třída obstarávající komunikaci s databází. Součástí je také vygenerovaný SQL script pro vytvoření databáze. SQL script obsahuje jednotlivě vygenerované tabulky, které odpovídají entitní vrstvě. SQL script zahrnuje i vytvoření vzájemných vztahů mezi tabulkami.

5.2.8 Generování PNG obrázků

Pro samotný export namodelovaného diagramu do PNG obrázku jsou v aplikaci vytvořeny tři funkce. V inicializační funkci je nastavena událost, která detekuje kliknutí na konkrétní HTML odkaz. Při kliknutí na tento HTML odkaz je volána funkce dlCanvas. Funkce dlCanvas vytvoří nový dočasný canvas element, který slouží pouze pro jednorázové vykreslení prvků. Nejdříve jsou všechny prvky (elementy a relace) přesunuty o určitou vzdálenost. Tím se vyplní prázdný prostor, který na našem pracovním canvasu zabírají příslušné panely. Prvky jsou poté vykresleny na náš nový dočasný canvas, a z tohoto canvasu je získán datový řetězec, který reprezentuje samotný PNG obrázek. Takto vytvořený obrázek je následně nabídnut uživateli ke stažení. Na závěr je dočasný canvas odstraněn a je zavolána funkce MoveBackPos. Tato funkce přesouvá všechny

prvky na původní pozici. Třetí funkce downloadCanvas simuluje kliknutí na HTML odkaz, tím se vyvolá již zmíněná metoda dlCanvas.

5.2.9 Ukládání a načítání diagramu

Vytvořené diagramy je možné uložit a zpětně načíst. K tomu slouží funkce saveDiagram a loadDiagram. Uložit diagram je možné dvěma způsoby. V případě jakéhokoliv neúspěchu je uživatel informován o chybě.

- saveDiagram - Funkce nejdříve projde v cyklu všechny elementy a relace. Ty pak převede na formát, který je možné uložit jako text. Odstraněny jsou veškeré reference a vnořené objekty, které jsou nahrazeny jednoznačnými textovými hodnotami. Pomocí JSONu jsou pak převedeny na jednotný formát. Funkce má dva parametry. Prvním parametrem je název, pod kterým je diagram uložen a druhý parametr obsahuje způsob uložení. V případě prvního způsobu uložení je diagram uložen do paměti prohlížeče (local browser storage). Druhý způsob umožňuje vytvoření textového souboru, ve kterém je stejným způsobem zapsán řetězec v JSON formátu. Tento soubor je pak nabídnut uživateli ke stažení.
- loadDiagram - Funkce načítá diagram. Má dva parametry, data a způsob načtení. V případě prvního způsobu data obsahují název načítaného diagramu. Z lokální paměti prohlížeče je digram načten a převeden z JSON formátu. V případě druhého způsobu obsahují data již přímo JSON řetězec, který uživatel načetl z dříve vytvořeného souboru. Následně jsou všechny objekty znovu vytvořeny tak, aby odpovídali specifikaci našich objektů. Jsou znovu vytvořeny reference i nastaveny správné hodnoty všem vnořeným objektům.

5.2.10 Ostatní funkce

V rámci aplikace vznikla celá řada dalších funkcí reprezentující drobnou funkcionalitu, kterou není nutné podrobně rozepisovat. Zde si je jen krátce představíme.

- mouseDown – Detekuje událost, při které uživatel zmáčkne tlačítko myši. Pokud se kurzor nachází v oblasti horního panelu, volá funkci, která odpovídá patřičnému tlačítku. V případě, že se kurzor nachází v levém panelu, umožňuje táhnout ikonu elementu nebo relace na kreslící plátno. Dále kontroluje, zda nebyl vybrán některý z objektů. To je detailně rozebrano v kapitole 5.3.2
- mouseMove – Detekuje pohyb kurzoru. V případě, že je vybrán element a uživatel najede na jeden z uzlů, které mění velikost, změní se i vzhled kurzoru. V kombinaci s mouseDown umožňuje měnit velikost elementu nebo přesouvat relaci.
- doubleClick – Detekuje dvojklik myši a používá se výhradně pro editaci částí elementů. Editace elementů je detailně popsána v kapitole 5.3.1.

- mouseUp – Detekuje událost, při které uživatel pustí tlačítko myši. Resetuje několik proměnných jako například tažení ikonky. Pokud byla vybrána relace, pak tato funkce dále kontroluje, zda byla relace připojena k jednomu z elementu.
- Funkce, kde název končí na HTML – Jedná se o funkce, které dynamicky mění obsah HTML stránky nebo které získávají hodnoty z HTML. Typickým příkladem je načtení diagramu z lokálního uložiště prohlížeče, kdy se zobrazí seznam uložených diagramů.
- Invalidate – Funkce, která nastavuje jednu boolean hodnotu na false. Pokud je tato hodnota nastavena na false, dojde při dalším volání metody mainDraw k překreslení. Funkce mainDraw je popsána v kapitole 5.2.2. Tato funkce je volána vždy, když provedeme nějakou změnu, kterou je nutné promítnout na canvas.

5.3 Problémy při implementaci a popis řešení

Při implementaci webové aplikace nastalo několik problémů, které nebyly popsány v návrhu. Menší implementační problémy byly vysvětleny již v kapitole 5.2. Zde si vysvětlíme pár příkladů, které bylo potřeba vyřešit důmyslněji.

5.3.1 Editace části elementů

K editaci jednotlivých částí slouží hned několik funkcí a jedná se o hybridní řešení, které využívá možnosti HTML, CSS a JavaScriptu. V HTML souboru je od začátku umístěna textová oblast (textarea). V inicializační funkci je na tuto oblast získána reference a ta je udržována v globální proměnné. Textarea je pomocí kaskádových stylů nastavená tak, aby nebyla viditelná. První funkce doubleClick detekuje událost, při které udělá uživatel dvojklik. Pokud máme vybraný některý z elementů, pak tato událost zavolá funkci setAreaPosAndResize. Této funkci dále předá v parametrech, do jaké části elementu uživatel dvojklikl a také jaký element byl vybrán. Současně je ale do textové oblasti vepsána hodnota té části elementu, kde uživatel klikl. Pokud jsme tedy dvojklikli na úpravu atributů, jsou všechny atributy zkopírovány do textové oblasti. Funkce setAreaPosAndResize má v sobě zabudovaný switch. Ten podle patřičné oblasti pomocí reference na textovou oblast nastaví její výšku, šířku a pozici. Díky tomu se při dvojkliku umístí textová oblast na požadované místo v elementu (Obrázek 14).

Pokud je textová oblast zobrazena, je možné v ní editovat všechny záznamy jako prostý text. Pokud vybereme jiný element nebo klikneme mimo současný element, je textová oblast zase schována a její hodnoty jsou přepsány do objektu samotného elementu. Toto vykonává funkce getDataFromArea. Zobrazování a skrývání textové oblasti obstarávají metody showArea a hideArea, které jednoduše mění styl zobrazení (Obrázek 14).

5.3.2 Výběr objektů na canvasu

Druhým zásadním problémem byl výběr elementů a relací na canvasu. Protože canvas nemá žádnou zabudovanou možnost jak detekovat, který objekt byl vybrán, bylo nutné tuto funkcionalitu naimplementovat. Jako řešení se nabízelo porovnávat oblast, kde uživatel klikl se všemi vytvořenými objekty, tedy jejich pozicemi. To se ukázalo jako značně nepraktické zvláště při porovnávání relací, kde nebylo možné pomocí souřadnic jednoznačně vymezit prostor, který spadá pod relaci.

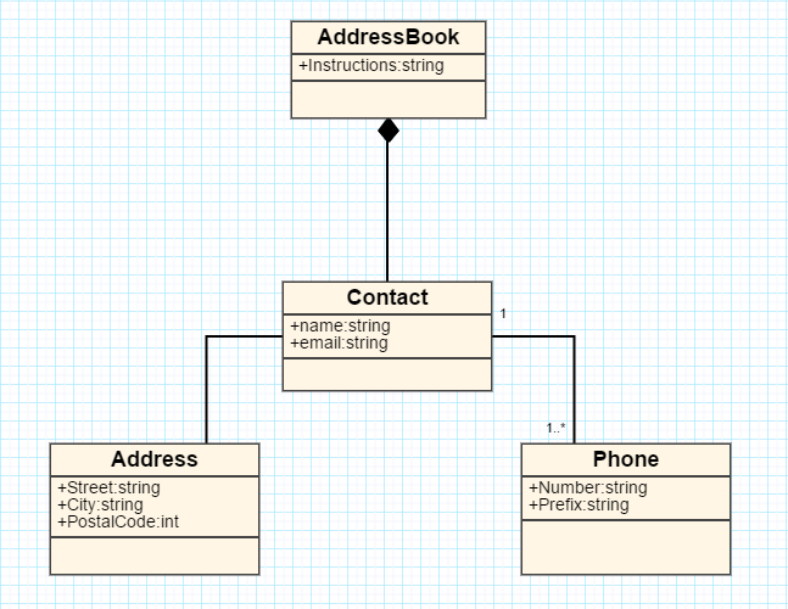
Řešení tohoto problému spočívalo ve vytvoření GhostCanvasu. V inicializační funkci je dynamicky na pozadí vytvořen další canvas. Ten není pro uživatele viditelný. Při kliknutí je volána metoda mouseDown. Ta detekuje, na kterých souřadnicích uživatel kliknul. Následně jsou všechny elementy a relace postupně vykreslovány na GhostCanvas. Proto jsme předávali u všech vykreslovacích funkcí context jako parametr, tím je možné použít vykreslovací funkce jak při klasickém vykreslování, tak při vykreslování na GhostCanvas. Vykreslován je vždy jen jeden objekt v jedné iteraci cyklu, současně je tedy na GhostCanvasu pouze jeden objekt. Na GhostCanvas vykreslujeme stejně jako na náš canvas, jediný rozdíl je v tom, že používáme pouze černou barvu výplně. Následně na GhostCanvasu voláme metodu getImageData se souřadnicemi uživatelského kurzoru a velikosti jednoho pixelu. Na závěr nám stačí jen zkontrolovat, jestli mají vrácená data větší hodnotu než nula. Jestliže je hodnota větší než nula, znamená to, že uživatel ukazuje na náš právě vykreslený objekt a ten můžeme označit jako vybraný objekt. Pokud se tato hodnota rovná nule, znamená to, že uživatel ukazuje na transparentní, nevybarvené místo a vykreslený objekt v této iteraci není objektem, na který uživatel ukazuje. To ale neznamená, že uživatel na žádný objekt neukazuje (uživatel může ukazovat na jiný objekt, který bude testován například až v další iteraci).

6 Testování vygenerovaných zdrojových kódů

Stejně jako aplikace byly i vygenerované zdrojové kódy postupně testovány, tím se v generovacích funkcích vyladili jednotlivé nepřesnosti a také se ověřila funkčnost vygenerovaných kódů. Vygenerované kódy je možné okamžitě napojit na GUI. V této krátké kapitole si rozebereme používání vygenerovaných zdrojových kódů na konkrétním namodelovaném diagramu.

6.1 Použitý testovací diagram

Pro účely testování byl zvolen jednoduchý diagram namodelovaný v naší aplikaci, který reprezentuje seznam kontaktů. První třída AdressBook reprezentuje klasický seznam, ve kterém jsou uloženy instance třídy Contact. Třída Contact dále obsahuje další instance třídy Address a Phone (Obrázek 25).



Obrázek 25: Testovaný diagram

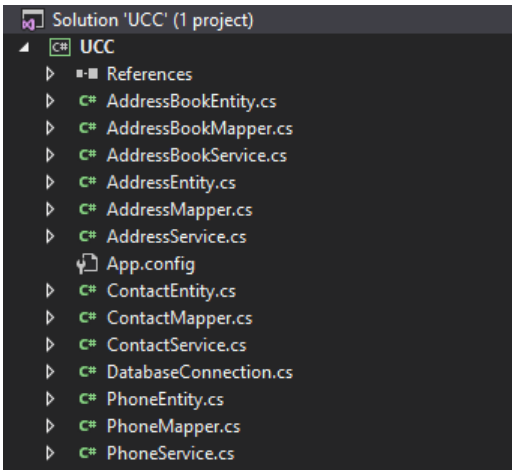
6.2 Struktura vygenerovaných kódů

Pokud uživatel zvolí možnost vygenerování zdrojového projektu do VS spolu s databází, jsou pro každou z těchto tříd vygenerovány tři vrstvy (Obrázek 26):

- Vrstva Entity představuje objektovou reprezentaci databázových záznamů. Jsou zde uloženy veškeré namodelované atributy jako property (vlastnosti).
- Vrstva Mapper komunikuje se Třídou DatabaseConnection. Jsou zde vytvořeny základní dotazy na databázi.

- Vrstva Service zabalí funkce mapperu. Požadované funkce jsou pak volány přímo na instancích této třídy a ne na instancích mapperu.

Současně je vygenerován i zdrojový kód na vytvoření databáze. Konkrétní detaily generování jsou popsány v implementační kapitole.



Obrázek 26: Vygenerované vrstvy a třída pro komunikaci s databází

6.3 Použití vytvořených zdrojových kódů

Celý vygenerovaný projekt jde po vytvoření statické metody Main ihned zkompilovat. Dalším požadavkem je nastavení připojovacího řetězce k databázi. V tomto řetězci je nutné specifikovat přihlašovací jméno, heslo a adresu SQL serveru. Řetězec se nachází ve třídě DatabaConnection.

Pro jednoduchost budou všechny příklady popsány nad třídou a tabulkou AdressBook. Do tabulky AdressBook byly přidány vzorové záznamy. Jak manipulovat s těmito záznamy si ukážeme v následujících podkapitolách.

idAddressBook	Instruction
1	Friends
2	High school
3	Work

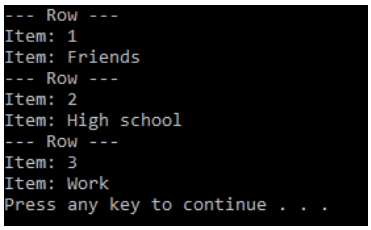
Obrázek 27: Vzorové data v tabulce AddressBook

Všechny funkce byly testovány v konzolové aplikaci, nicméně způsob napojení vygenerovaných kódů na výstupní technologii je zcela v režii programátora.

6.3.1 Použití funkce SelectAll

Funkce SelectAll poskytuje možnost vypsání všech záznamů z dané databázové tabulky. Nejdříve je nutné vytvořit instanci třídy AdressBookService a instanci třídy DataTable. Následně je nad vytvořeným objektem třídy AdressBookService zavolána již výše zmíněná metoda SelectAll. Tato

metoda vrací objekt typu DataTable. Všechny záznamy jsou poté vypsány do konzole použitím cyklu foreach (Obrázek 28). Tyto vypsané záznamy se shodují se vzorovými daty. (Obrázek 27).



Obrázek 28: Výpis při použití funkce SelectAll

6.3.2 Použití funkce Insert

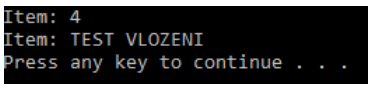
Jak už název napovídá, funkce Insert má za úkol vložit záznam do příslušné tabulky. Nejdříve je nutné vytvořit instance třídy AdressBookService a AdressBookEntity. Naše třída AdressBook měla pouze jeden atribut Instructions. U našeho objektu třídy AdressBookService tuto property nastavíme a následně zavoláme požadovanou metodu Insert. Metoda Insert vrací jedničku v případě úspěchu a nulu v případě neúspěchu. Pro jistotu můžeme tuto hodnotu ještě vypsat do konzole. Záznam je tímto vložen do tabulky a při novém volání metody SelectAll obsahuje tabulka již 4 záznamy (Obrázek 29).

idAddressBook	Instruction
1	Friends
2	High school
3	Work
4	TEST VLOZENI

Obrázek 29: Upravená vzorová data v tabulce AddressBook

6.3.3 Použití funkce SelectById

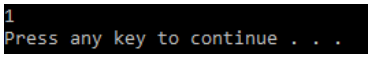
Tato funkce vybírá konkrétní záznam podle ID. Stejně jako u funkce SelectAll, musíme pro použití této funkce vytvořit instance třídy AddressBookService a DataTable. Následně musíme nad objektem třídy AddressBookService zavolat funkci SelectById, této funkci předáváme v jediném parametru ID požadovaného záznamu. Výsledek je opět vrácen do DataTable objektu a tento výsledek je následně také vypsán. (Obrázek 30).



Obrázek 30: Výpis konkrétního záznamu do konzole

6.3.4 Použití funkce Delete

Funkce Delete vymaže jediný záznam podle konkrétního ID. Pro použití musíme nejdříve vytvořit instanci třídy AddressBookService a také musíme vytvořit kontrolní proměnnou. Následně nad naším objektem třídy AddressBookService zavoláme funkci Delete s ID záznamu, který chceme vymazat. Stejně jako funkce Insert, i funkce Delete vrací jedničku v případě úspěchu a nulu v případě neúspěchu. Výsledek funkce Delete uložíme do kontrolní proměnné a tu můžeme následně vypsat (Obrázek 31).



Obrázek 31: Výpis úspěšného vymazání záznamu z tabulky

6.3.5 Použití funkce Update

Funkce Update upraví konkrétní záznam podle předaných nových parametrů. Nejdříve je nutné vytvořit instance třídy AddressBookService a AddressBookEntity. Následně musíme u objektu třídy AddressBookEntity nastavit property Instructions. Stejně jako v předchozích případech je zde pro kontrolu vytvořena proměnná, do které ukládáme vrácený výsledek funkce. Poté je nad objektem třídy AddressBookService zavolána funkce Update. Této funkci předáváme vytvořený entitní objekt a ID záznamu, který chceme upravit. Výsledná tabulka následně ukazuje 3 záznamy (Obrázek 32).

idAddressBook	Instruction
1	Friends
3	Work
4	University

Obrázek 32: Tabulka nad třídou AdressBook, po editaci testovacího záznamu

Oproti předchozí ukázce (Obrázek 29) obsahuje poslední tabulka několik změn. Záznam s ID 2 byl v podkapitole 6.3.4 vymazán a přidaný testovací záznam byl v této podkapitole editován.

7 Závěr

Cílem této bakalářské práce bylo navrhnout, analyzovat a vytvořit webovou aplikaci pro tvorbu třídních diagramů, která umožní generování entitní a databázové vrstvy. Souběžným cílem bylo také nastudovat a popsat problematiku tvorby třídních diagramů a následně získané informace promítnout do implementace aplikace. V teoretické částí práce byla stručně popsána problematika tvorby třídních diagramů a jejich převodů na zdrojové kódy. Následně byla provedena důkladná analýza, ze které vznikl první návrh webové aplikace. Tento návrh byl poté promítnut do samotného řešení aplikace. Práce zahrnuje také uživatelskou dokumentaci, popis implementace a testování vygenerovaných zdrojových kódů.

Výsledkem této práce je webová aplikace s jednoduchým a intuitivním rozhraním, která umožňuje tvorbu třídních diagramů. Tyto diagramy jde vytvořit, uložit nebo načíst a následně editovat. Aplikace také umožňuje generovat zdrojové kódy do C# jazyka. Takto vygenerované zdrojové kódy jsou přizpůsobeny volbě uživatele. Na základě této volby vygenerovaný archiv obsahuje pouze zdrojové kódy nebo projekt spustitelný ve Visual Studiu. K vygenerovaným zdrojovým kódům je možné také zahrnout SQL script pro vytvoření databáze. V tomto případě jsou z každé diagramové třídy vygenerovány tři programové třídy reprezentující jednotlivé vrstvy. Tyto vrstvy následně obsahují všechny potřebné části pro práci s databází, včetně naimplementovaných základních operací. Mezi tyto základní operace patří výběr záznamu podle ID, výběr všech záznamů, vložení záznamu, úprava a smazání záznamu. Cíle této bakalářské práce byly splněny v plném rozsahu s několika drobnými rozšířeními, které značně usnadňují práci se samotnou aplikací.

Při implementaci této webové technologie jsem se seznámil s některými dosud nepoznanými technologiemi. V návrhové části jsem si značně prohloubil znalosti UML jazyka a také jsem získal praktický pohled na návrh a analýzu aplikace. Implementační část mě seznámila s možnostmi HTML5 canvasu a při tvorbě samotné aplikace jsem si značně zdokonalil JavaScriptové znalosti. I přes velké množství času, který jsem věnoval této bakalářské práci, hodnotím tuto zkušenost jako velice přínosnou.

Webovou aplikaci plánuji dále rozšiřovat vytvořením serverové části v NodeJS. K tomu bude zapotřebí provést revizi napsaného kódu. To mi poskytne možnost přesunout část logiky z klientské aplikace na serverovou část. V serverové části plánuji vytvořit systém přihlašování, který poskytne uživateli větší podporu při modelování diagramů. Uživatelé následně budou moci zakládat vývojářské týmy, kde budou uživatelé schopni přistupovat k namodelovaným diagramům současně. Také zde plánuji vytvoření úvodní nabídky a vylepšení některých současných funkcí. Vylepšené grafické rozhraní by také mělo nabízet podporu jednotlivých záložek, mezi kterými půjde jednoduše přepínat namodelované diagramy. Nevylučuji ani podporu dalších programovacích jazyků a databází nebo dalších UML diagramů. Spolu s těmito dokončenými funkcemi by se následně aplikace mohla zařadit mezi konkureční freewarové nástroje, umožňující vytváření třídních (nebo jiných) UML diagramů.

Literatura

[1] ARLOW J., NEUSTADT I. : UML 2 a unifikovaný proces vývoje aplikací. Dotisk prvního vyd. Computer Press, a.s. , 2011. ISBN 978-80-251-1503-9.

[2] UML 2.4 Diagrams Overview[online]. [cit. 2016-04-06].
Available from WWW: http://www.uml-diagrams.org/uml-24-diagrams.html

[3] Class and Object Diagrams Overview [online]. [cit. 2016-04-06].
Available from WWW: http://www.uml-diagrams.org/class-diagrams-overview.html

[4] UML - Class diagram [online]. [cit. 2016-04-06]. Dostupné z WWW:
http://www.itnetwork.cz/navrhove-vzory/uml/uml-class-diagram-tridni-model/

[5] UML Attribute [online]. [cit. 2016-04-06]. Available from WWW:
http://www.uml-diagrams.org/property.html#classifier-attribute

[6] UML Operation [online]. [cit. 2016-04-06]. Available from WWW:
http://www.uml-diagrams.org/operation.html

[7] VONDRÁK, I., KOŽUSZNIK, J., OCHODKOVÁ, E.: Metody specifikace softwarových systémů. VŠB-TUO, Česká Republika, Ostrava, 2006

[8] UML2 Class Diagram in Java [online]. 31.12.2012 [cit. 2016-04-06]. Available from WWW:
https://dzone.com/articles/uml2-class-diagram-java

[9] Mapping objects to relational databases [online]. 1.7.2000 [cit. 2016-04-06]. Available from WWW: https://www.ibm.com/developerworks/library/ws-mapping-to-rdb/#h1

[10] FOWLER, M., RICE, D., FOEMMEL, M., HIEATT, E., MEE, R.:
Randy StaffordPatterns of Enterprise Application Architecture 2002. ISBN 978-80-251-1503-9.

[11] Selectable Shapes Part 2: Resizable, Movable Shapes on HTML5 Canvas [online]. [cit. 2016-04-06]. Available from WWW: http://simonsarris.com/blog/225-canvas-selecting-resizing-shape

A Datové CD

Součástí bakalářské práce je přiložené CD s následující adresářovou strukturou:

- /Thesis - Obsahuje vypracovaný text bakalářské práce ve formátu PDF.
- /Application - Obsahuje celou aplikaci.

Adresář Application dále obsahuje:

- /css - Adresář pro css soubory.
- /image - Adresář pro obrázky, které používá aplikace.
- /javascript - Adresář, který obsahuje JavaScriptové soubory.
- /javascript/shapes.js - JavaScriptový soubor obsahující hlavní logiku aplikace.
- /javascript/MySQL.js - JavaScriptový soubor obsahující konstanty pro MySQL.
- /javascript/MSSQLS8.js - JavaScriptový soubor obsahující konstanty pro MS SQL 2008.
- /javascript/CsharpConst.js - JavaScriptový soubor obsahující konstanty pro C# jazyk.
- /javascript/FileSaver.js - Externí knihovna pro ukládání dat.
- /javascript/jszip.js - Externí knihovna pro vytváření ZIP archívu.
- /javascript/boxes2.js - Původní JavaScriptový soubor pro přesouvání objektů na canvasu, který je popsán v kapitole 5.2.
- /index.html - HTML soubor spouštějící aplikaci.